Avaya™ Interaction Center

Release 6.0

Agent Data Unit Server Programmer's Guide

DXX-1015-01
Issue 1.0
June 2002

Notice
Every effort was made to ensure that the information in this book was complete and accurate at the time of printing. However, information is subject to change.

Preventing Toll Fraud
"Toll fraud" is the unauthorized use of your telecommunications system by an unauthorized party (for example, a person who is not a corporate employee, agent, subcontractor, or working on your company's behalf). Be aware that there may be a risk of toll fraud associated with your system and that, if toll fraud occurs, it can result in substantial additional charges for your telecommunications services.

Avaya Fraud Intervention
If you *suspect that you are being victimized* by toll fraud and you need technical support or assistance, call Technical Service Center Toll Fraud Intervention Hotline at +1 800 643 2353.

Providing Telecommunications Security
Telecommunications security (of voice, data, and/or video communications) is the prevention of any type of intrusion to (that is, either unauthorized or malicious access to or use of your company's telecommunications equipment) by some party.

Your company's "telecommunications equipment" includes both this Avaya product and any other voice/data/video equipment that could be accessed via this Avaya product (that is, "networked equipment").

An "outside party" is anyone who is not a corporate employee, agent, subcontractor, or working on your company's behalf. Whereas, a "malicious party" is anyone (including someone who may be otherwise authorized) who accesses your telecommunications equipment with either malicious or mischievous intent.

Such intrusions may be either to/through synchronous (time-multiplexed and/or circuit-based) or asynchronous (character-, message-, or packet-based) equipment or interfaces for reasons of:

- Utilization (of capabilities special to the accessed equipment)
- Theft (such as, of intellectual property, financial assets, or toll-facility access)
- Eavesdropping (privacy invasions to humans)
- Mischief (troubling, but apparently innocuous, tampering)
- Harm (such as harmful tampering, data loss or alteration, regardless of motive or intent)

Be aware that there may be a risk of unauthorized intrusions associated with your system and/or its networked equipment. Also realize that, if such an intrusion should occur, it could result in a variety of losses to your company (including but not limited to, human/data privacy, intellectual property, material assets, financial resources, labor costs, and/or legal costs).

Your Responsibility for Your Company's Telecommunications Security
The final responsibility for securing both this system and its networked equipment rests with you - an Avaya customer's system administrator, your telecommunications peers, and your managers. Base the fulfillment of your responsibility on acquired knowledge and resources from a variety of sources including but not limited to:

- Installation documents
- System administration documents
- Security documents
- Hardware-/software-based security tools
- Shared information between you and your peers
- Telecommunications security experts

To prevent intrusions to your telecommunications equipment, you and your peers should carefully program and configure your:

- Avaya-provided telecommunications systems and their interfaces
- Avaya-provided software applications, as well as their underlying hardware/software platforms and interfaces
- Any other equipment networked to your Avaya products.

Avaya National Customer Care Center
Avaya provides a telephone number for you to use to report problems or to ask questions about your contact center. The support telephone number is 1-800-242-2121.

Ordering Information
Avaya Publications Center
Voice: +1 800 457 1235
International Voice: 410 568 3680
Fax: +1 800 457 1764
International Fax: 410 891 0207
Email: totalware@gwsmail.com

Write: GlobalWare Solutions
Attention: Avaya Account Manager
200 Ward Hill Avenue
Haverhill, MA 01835 USA

Order: Document No. DXX-1015-01, Issue 1.0, June 2002

To order product documentation online, go to http://www.avayadocs.com, click on Online Services, and select the appropriate product group.

Warranty
Avaya Inc. provides a limited warranty on this product. Refer to the "Limited Use Software License Agreement" or other applicable documentation provided with your package to establish the terms of the limited warranty.

Avaya Web Page
http://www.avaya.com

Trademarks
Avaya, Conversant, CustomerQ, Definity, DefinityOne, Nabnasset, Quintus, and WebQ are registered trademarks or trademarks of Avaya Inc. in the United States or other countries or both.

Portions of Avaya Interaction Center include technology used under license as listed below, and are copyright of the respective companies and/or their licensors:

ActivePerl is a trademark of ActiveState Tool Corp. This product includes software developed by the Apache Software Foundation (http://www.apache.org/). Cognos, Impromptu and Powerplay are registered trademarks of Cognos Incorporated. YACC++ is a registered trademark of Compiler Resources, Inc. APEX, ComponentOne, VideoSoft, True DBGrid, VSVIEW, SizerOne, VS-OCX, VSFlexGrid, VSFORUM, VSREPORTS, VSDOCX, VSSPELL, and TrueDBList are either registered trademarks or trademarks of ComponentOne LLC. CT Connect, Dialogic, Intel, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. Hummingbird is a registered trademark of Hummingbird, Ltd. SearchServer is a trademark of Hummingbird, Ltd. RISC System/6000 and DirectTalk/2 are trademarks of International Business Machines Corporation in the United States or other countries or both. IBM, OS/2, AS/400, CICS, WebSphere, CT, VisualAge, and DirectTalk are registered trademarks of International Business Machines Corporation in the United States or other countries or both. Lotus and Lotus Sametime are trademarks or registered trademarks of Lotus Development Corporation and/or IBM Corporation in the United States, other countries, or both. VisualX is a registered trademark of Intergroup Technologies, Inc. ActiveX, Visio, Internet Explorer, Windows, Windows NT, Windows 2000, Win32s, SQL Server, Visual Basic, Visual C++, Outlook, and FrontPage are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. TimesTen is a registered trademark of TimesTen Performance Software. Oracle is a registered trademark, and Oracle8i and Oracle® SQL/Services are trademarks or registered trademarks of Oracle Corporation. Rogue Wave and .h++ are registered trademarks of Rogue Wave Software Inc. SourcePro is a trademark of Rogue Wave Software, Inc. Siebel is a trademark of Siebel Systems, Inc. BasicScript is a registered trademark of Summit Software Company. Sun, iPlanet, Java, Solaris JRE, J2EE, JavaServer Pages, and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. SPARC is a registered trademark of SPARC International, Inc. Products bearing SPARC trademarks are based on an architecture developed by Sun Microsystems, Inc. In3D is a trademark of Visual Insights, Inc. InstallShield® is a registered trademark and service mark of InstallShield Software Corporation in the United States and/or other countries. ORBacus is a trademark of IONA Technologies PLC. Formula One is a licensed trademark and Tidestone Technologies, Inc. Visual Components, First Impression, and VisualSpeller are registered trademarks of Tidestone Technologies, Inc. JRun is a trademark of Macromedia, Inc. in the United States and/or other countries. Intervoice is a registered trademark of Intervoice-Brite, Inc. UNIX is a registered trademark of The Open Group in the United States and other countries. Acrobat is a registered trademark of Adobe Systems.

Other product and brand names are trademarks of their respective owners.

Acknowledgment
This document was written by the CRM Information Development group of Avaya

CONTENTS

BEFORE YOU BEGIN

Typographical Conventions

This guide uses the following font conventions:

Font Type	Meaning
`code`	This font signifies commands, information that you enter into the computer, or information contained in a file on your computer.
italics	This font is used to add emphasis to important words and for references to other chapter names and manual titles. It also indicates variables in a command string.
jump	Blue text in online documents indicates a hypertext jump to related information. To view the related material, click on the blue text.

Notes, Tips, and Cautions

Note: A note calls attention to important information.

Tip: A tip offers additional how-to advice.

Caution: A caution points out actions that may lead to data loss or other serious problems.

Contacting Technical Support

If you are having trouble using Avaya software, you should:

1. Retry the action. Carefully follow the instructions in written or online documentation.
2. Check the documentation that came with your hardware for maintenance or hardware-related issues.

3 Note the sequence of events that led to the problem and the exact messages displayed. Have the Avaya documentation available.

4 If you continue to have a problem, contact Avaya Technical Support by:

- Logging in to the Avaya Technical Support Web site (http://www.avaya.com/support/qq).
- Calling or faxing one of the following numbers from 8:30 a.m. to 8:30 p.m. (Eastern Standard Time), Monday through Friday (excluding holidays):
  - Toll free in the U.S. only: 1-888-TECH-SPT (1-888-832-4778)
  - Direct line for international and domestic calls: 512-425-2201
  - Direct line for faxes: 512-719-8225
- Sending email with your question or problem to `crmsupport@avaya.com`. You may be asked to email one or more files to Technical Support for analysis of your application and its environment.

Note: If you have difficulty reaching Avaya Technical Support through the above URL or email address, please go to www.avaya.com for further information.

Product Documentation

Most Avaya product documentation is available in both printed and online form. However, some reference material is available only online, and certain information is available only in printed form. A PDF document with detailed information about all of the documentation for the Avaya Interaction Center is included in the `Doc` directory on the product CD-ROM. This PDF document is also included on the separate documentation CD-ROM.

Readme File

The Readme file is an HTML file included on the Avaya Interaction Center software CD-ROM. This file contains important information that was collected too late for inclusion in the printed documentation. The Readme file can include installation instructions, system requirements, information on new product features and enhancements, suggested work-arounds to known problems, and other information critical to successfully installing and using your Avaya software. You may also receive a printed Addendum to the Readme, containing similar information uncovered after the manufacture of the product CD-ROM. You should review the Readme file and the Readme Addendum before you install your new Avaya software.

Electronic Documentation

The electronic documentation (in PDF or HTML format) for each Avaya Interaction Center product is installed automatically with the program. Electronic documentation for the entire Avaya product suite is included on the product CD-ROM and the documentation CD-ROM.

You can also view the documentation set online at http://www.avayadocs.com.

Printed Documentation

You can purchase printed copies of these manuals separately. For details, see on the back of this manual's title page.

License to Print the Electronic Documentation

Online copies of documentation are included on the CD-ROM that accompanies every software release. An Avaya customer who has licensed software (a "Licensee") is entitled to make this online documentation available on an internal network or "intranet" solely for the Licensee's use for internal business purposes. Licensees are granted the right to print the documentation corresponding to the software they have purchased solely for such purposes.

Right-To-Print License Terms

Documents must be printed "as-is" from the provided online versions. Making changes to documents is not permitted. Documents may be printed only by any employee or contractor of Licensee that has been given access to the online documentation versions solely for Licensee's internal business purposes and subject to all applicable license agreements with Avaya. Both online and printed versions of the documents may not be distributed outside of Licensee enterprise or used as part of commercial time-sharing, rental, outsourcing, or service bureau use, or to train persons other than Licensee's employees and contractors for Licensee's internal business purposes, unless previously agreed to in writing by Avaya. If Licensee reproduces copies of printed documents for Licensee's internal business purposes, then these copies should be marked "For internal use only within <Licensee> only." on the first page or cover (where <Licensee> is the name of Licensee). Licensee must fully and faithfully reproduce any proprietary notices contained in the documentation. The copyrights to all documentation provided by Avaya are owned by Avaya and its licensors. By printing any copy of online documentation Licensee indicates its acceptance of these terms and conditions. This license only governs terms and conditions of printing online documentation. Please reference the appropriate license agreement for terms and conditions applicable to any other use, reproduction, modification, distribution or display of Avaya software and documentation.

Educational Services

Avaya University provides excellent training courses on a variety of topics. For the latest course descriptions, schedules, and online registration, you can get in touch with us:

- Through the web at http://learning2.avaya.com
- Over the telephone at 800-288-5327 (within the U.S.) +001 303-406-6089 (outside of the U.S.)
- Through email at `Avaya.U.Helpdesk@accenture.com`

CHAPTER 1
THE ADU SERVER

Overview

The Agent Data Unit (ADU) Server is responsible for tracking the state of agents at the contact center. Agents are also referred to as customer service representatives, or CSRs. The term “agent” is used throughout this book to signify a customer service representative.

When an agent logs in the Avaya Interaction Center (Avaya IC), the ADU Server creates a record of the agent's session on the system. This record is called an Agent Data Unit or ADU. The ADU contains information such as the agent’s login ID, equipment number, phone type, time of login, state (InCall, WrapUp, and so on), and the time the current agent state was entered. It also contains a unique identifier (ADUID) for each ADU. The information in an ADU is used by IC Manager to generate real-time Agent Status monitors.

The ADU Server manages the ADU throughout its lifecycle. It creates new ADUs, stores open ADUs, and provides services that enable clients to interact with an agent's record. When an agent ends a session on the system, the server terminates the agent’s ADU.

The ADU Server is also responsible for monitoring the contents of ADUs. When ADUs are modified, the ADU Server generates events to interested clients.

Note: The ADU server is vary similar in function to the EDU (Electronic Data Unit) Server. With minor exceptions, the ADU Server functions the same way as the EDU Server.

Start-up Procedures

When Avaya IC starts up, the Avaya Toolkit starts up the ADU Server. When an agent logs in to Avaya IC, the Toolkit invokes the ADU.FindOr() method, causing the ADU Server to find or create an ADU for the agent.

If for some reason the ADU Server is not running when an agent logs in, the ADU.Create() method invocation causes the ADU Server to be started.

Cooperation of ADU Servers

When a new ADU Server is added to Avaya IC, existing ADU Servers must be made aware of the new server through use of the IC Manager. Refer to *IC Administration Volume 1: Servers & Domains* for information on updating servers.

When the ADU Servers have been made aware of each other, they can make requests of each other. This is most significant in the area of the Assign method. When a client Assigns to an ADU Server, the Assign is relayed to other ADU Servers, and they watch for ADUs that match the client's Assign criteria as well. This makes Assign a relatively expensive operation, and clients should be designed so they only need to Assign once to specify the ADUs in which they are interested. Assigning to multiple ADU Servers is unnecessary and causes numerous problems.

Every ADUID identifies the server that created it and the time at which it was created. When a client requests access to an ADU, the ADU Server that the client is using acts as a proxy to the originating ADU Server of the ADU.

CHAPTER 2
THE AGENT DATA UNIT

Definition of an ADU

When an agent logs in to Avaya IC for the first time, a record called an Agent Data Unit (ADU) is created. The ADU contains information about the agent's activities during the session. A typical ADU might contain the time the agent logged in, the various states the agent entered (InCall, WrapUp, and so on..), and the time when the agent last entered each state.

Note: When an agent logs back into Avaya IC, a new ADU is not created. Avaya IC uses the ADU that was created at the initial login.

The information in ADUs is used to create Agent Status monitors in IC Manager.

The ADU Lifecycle

The lifecycle of an ADU is comprised of three stages: creation, activity, and termination. Each of these stages is discussed in this section.

ADU Creation

Every agent session has a corresponding ADU. When an agent logs on to Avaya IC for the first time, the Avaya Toolkit invokes the ADU.Create() method, which passes the agent's Avaya IC login ID. In response, the ADU Server creates an ADU, assigns it a unique identifier called an ADUID, and stores the creation time and the agent's login ID in the ADU. This ADUID is used whenever the agent logs back into Avaya IC. In other words, when the same agent logs out and back in again, the ADU server will attempt to reuse the same ADUID.

This behavior differes slightly from the way ADUIDs were created in previous versions (prior to version 5.5). In older versions, a new ADUID was created each time an agent logged in.

ADU Activity

The ADU is a real-time storage record that collects strings of text from multiple sources. During its life, the dual job of the ADU is to log the agent's state information, as well as to collect the information ({name, value} data couples) entered by other clients or servers.

Any application that wants to interact with an ADU has to request it by its unique identifier, the ADUID. Upon request, the ADU Server passes the ADUID out to other processes, thus enabling applications to request access to individual ADUs.

The ADU Server also honors requests to watch for ADUs that contain certain values. If any ADU matches the monitoring criteria, the ADU Server issues event messages (which contain the ADUID) to "interested" clients. (Refer to "ADU Event Monitoring," on page 27.)

Typically, clients want to receive event notifications about ADUs, examine ADU contents, and modify ADU values. These activities are supported through method invocations. For example, a client could invoke one of the Get methods (GetOneValue(), GetSomeValues(), and so on) to ask the ADU Server to return values in an ADU. Chapter 7 of this manual describes the methods supporting these activities.

ADU Termination

When an agent logs out of Avaya IC, the agent's ADU should be closed.

Before an ADU can be closed, the ADU Server must first verify that all processes are done with it. For each ADU, the ADU Server maintains an internal list of processes that have read, transferred, or modified the ADU. Any process that invokes a method using the ADUID of the ADU is added to the list. When a client invokes the method ADU.Terminate(), the client's name is removed from the list. When the list is empty, the ADU can be terminated.

Note: In some configurations, it is possible for the agent to log out of the teleset without logging out of Avaya IC. The ADU is *not* closed in this case.

Safeguards are built into the ADU Server to terminate "suspicious" ADUs, in other words, those that appear to have been lost. Any ADU that has been idle for an extended period (thirty minutes, by default) is automatically terminated. (The idle time can be changed with the Idle Time configuration parameter, as described in "Configuration Parameters," on page 37.)

If the ADU Server goes down, all ADUs are closed. For Agent Status monitors to be accurate, all users must log off and log back on again to create new ADUs.

Listing Active ADUs

You can view a list of active ADUs and the clients interested in them with the listadu utility.

To use this utility:

1. Copy `listadu.exe` from the Avaya IC5.x/bin directory to the directory in which the `vesp.imp` and `vespidl.pk` files are located, which is /etc by default.

2. From the system prompt, type

```
listadu -iinterface -uusername -ppassword
```

-i interface to use (often ADU or VDU, or an alias or uuid of such a server)

-u user name to log in as (often Admin)

-p password of user

No space between -*x* and the text that follows.

Example:

```
listadu -iBostonADU -uAdmin -pDaphnie
```

There are defaults for all of these, but if a login error occurs, always specify -u and -p.

The ADUID

When an ADU is created, it is automatically given an identification number (ADUID) that uniquely defines the agent's Avaya IC session. The ADUID is a 32 byte hexadecimal string partially composed of the time and date of creation of the ADU, as well as its progenitor–the software process and machine that created it. A combination of the IP and port identifies which ADU Server is the progenitor of this ADU.

The following example illustrates the structure of an ADUID.

ADUID = 30f4216300000000780000261f430002

Bytes	Example Contents	Description
0–7	30f42163	Timestamp. Hex representation of the number of seconds since January 1, 1970.
8–11	0000	Hex representation of a uniqueness value
12–15	0000	Reserved
16–23	78000026	Hex representation of the IP address
24–27	1f43	Hex representation of a port
28–31	0002	Reserved

The ADU Data Table

The data in an ADU consists of a series of sequenced {name, value} couples that represent information about an agent's session on Avaya IC. For example, the name/value pair {**createtime**,19970704 08:00:01} represents the time the ADU was created. The field name **createtime** is the *name data element* and **19970704 08:00:01** is the *value assigned to the field*. Applications gain access to data by references to field names.

- Names are restricted to non-empty strings of less than 35 characters. (Container names may contain more than 35 characters. Refer to "Container Names and Special Tokens," on page 22 for information on container names.) Names may contain letters (a..z, A..Z), digits (0..9), and the underscore character (_). Data element names are case sensitive: "foo", "Foo", and "FOO" refer to three different data elements.

  The set of names that Avaya IC software reserves for its own system use are listed in the following section. Although they are not strictly reserved words in the technical sense, to use these field names for anything other than their pre-defined purposes could result in conflicts between applications.

- Values, the pieces of data assigned to a field, are strings that may contain from 0 to 4096 characters. Although longer strings are technically permissible, they are strongly discouraged. Any character except the null character ('\0') is legal. An average ADU is expected to contain approximately 100 values. It can hold many more values, but additional values require more memory, which can affect system performance, and each time the ADU Server must allocate additional memory the system slows briefly.

The ADU Server sets certain name/value data pairs, such as the ADUID. Other name/value data pairs are written into the ADU by other processes. Because the ADU can accept input from a variety of sources, data could come from other servers or the agent's application. Changes to all data values are recorded, including information on who made the change and when the change was made.

ADU Contents

This section describes the ADU fields for Avaya IC in three categories: Core, Agent, and Queue. The Core category contains the ADU fields that are used by agents and queues. The Agent and Queue categories provide the ADU fields used by agents and queues respectively.

The following information may be maintained in an ADU by various system processes. Not all of these items are maintained in all environments.

Caution: The field names listed below can be used in client applications. However, they should be treated just like reserved words with specific meanings. *Do not use these names for other than these explicit purposes!* Doing otherwise could result in misinterpretation. Some are read-only values to clients.

Core ADU Fields

The following ADU fields are used by both agents and queues on Avaya IC.

Field Name	Description	Set By
adu_id	A string that uniquely identifies the ADU.	ADU Server
createtime	The date and time the ADU was created in yyyy-mm-dd hh:mm:ss format.	ADU Server
createtimet	The time_t of the ADU creation.	ADU Server
duration	Historical field.	ADU Server
endtime	The time of the last successful termination of the ADU or equivalent such as removal due to idleness in hh:mm:ss format.	ADU Server
persistence	The date and time that the EDU was last stored in the DUStore Server.	EDU Server
suspend	The reason that the ADU was last removed from server memory. This can be any one of the following: ▪ normal–all involved parties have suspended use of the ADU. ▪ expired–removed from memory due to idleness. ▪ exit–server was shut down. ▪ overflow–removed from memory for other ADUs. ▪ forced–removed by ForceTerminate method.	ADU Server
termination	A string representing the reason for the termination of the ADU: normal, overflow, expired, or exit.	N/A
time	Reserved for internal use. This field is inserted into each event that is set to reporting packages and is reserved for that purpose.	N/A
type	The type of resource, set to “per”.	Tool Kit

Agent ADU Fields

The following ADU fields are used by agents on Avaya IC.

Field Name	**Description**	**Set By**
auxwork.<m>.detail	A code that describes the reason that the agent entered this state.	Blender Server
auxwork.<m>.endtime	The time that the agent exited this AuxWork state.	Blender Server
auxwork.<m>.starttime	The time that the agent entered the current AuxWork state. The <m> is the increment time that agent enters into a different AuxWork state.	Blender Server
ceiling	The maximum task load limit for the agent.	Workflow Server
cobject	A unique representation of the client's login.	Tool Kit
last_termination	The identity of the last user of the Terminate method, used as debugging information.	ADU Server
load	The maximum number of contacts that can be concurrently assigned to the agent across all of the media channels.	Workflow Server
login	The time that the agent logged in to Avaya IC in hh:mm:ss format.	Tool Kit
loginid	The agent's Avaya IC login id.	Tool Kit
logout	The time that the agent logged off from Avaya IC in hh:mm:ss format..	Tool Kit
<media>.abandoned	The total number of contacts that have been abandoned since the agent logged in to the media channel. (voice, email, web)	Media Connector Server
<media>.ceiling	The maximum task load limit on the media channel.	Workflow Server & IC Manager
<media>.connector	The UUID of the Media Connector Server: voice, email, or web.	Media Connector Server
<media>.contactcount	The number of contacts assigned to the agent for this media channel.	Media Connector Server
		(Sheet 1 of 4)

Field Name	Description	Set By
<media>.contactsoffered	The total number of contacts that arrived since the agent logged into this media channel. IC Manager computes handled contacts by taking the difference between <media>.contactsoffered and <media>.abandoned.	Media Connector Server
<media>.currentload	The current task load assigned to the agent for this media channel. This is used to throttle delivery of ADUs.	Workflow Server
<media>.load	The maximum number of contacts that can be concurrently assigned to the agent through this media channel.	Workflow Server & Blender Server
<media>.loginid	The login id of the agent for this media channel.	Media Connector Server
<media>.login	The time that the agent signed into the media channel in hh:mm:ss format.	Media Connector Server
<media>.logout	The time that the agent signed out of the media channel hh:mm:ss format.	Media Connector Server
<media>.privileges	This field defines the agent's capability to manipulate agent attributes, such as task load, for this media channel.	Media Connector Server & IC Manager
<media>.state	The media channel state of the agent. Normalized channel states are: ▪ Logged-In ▪ Logged-Out ▪ Active ▪ Available ▪ Busy ▪ Wrap-up	Workflow Server & Blender Server
<media>.state.<state>.total	The total duration time of the agent in the media channel state.	Media Connector Server
<media>.updatetime	The time that the media channel was last updated in hh:mm:ss format.	Media Connector Server
<media>.<n>.eduid	The Electronic Data Unit (EDU) associated with this contact where *n* is the contact number for the agent on this media channel.	Media Connector Server
<media>.<n>.sourcequeue	The queue from which the contact was received where *n* is the contact number for the agent on this media channel.	Media Connector Server
		(Sheet 2 of 4)

Field Name	Description	Set By
<media>.<n>.state	The current media contact state. Normalized contact states are: ▪ Alerting ▪ Active ▪ On-Hold ▪ Wrap-up ▪ Completed	Media Connector Server
<media>.<n>.state.<state>.starttime	The time that the media channel state started.	Media Connector Server
modifier	This field identifies the login id or uuid of the user who caused the event. It is inserted into each event that is sent to reporting packages.	ADU Server
owner	The loginid or UUID of the application that created the ADUID, or that last used the Transfer/SetTransfer methods. (historical).	ADU Server
privileges	This field defines the agent's capability to manipulate agent attributes, such as task load, for Avaya IC.	Workflow Server & IC Manager
state	The current agent state of the agent. Normalized agent states are: ▪ Logged-In ▪ Logged-Out ▪ Available ▪ InitAuxWork ▪ AuxWork	Workflow Server & Blender Server
statedetail	This field provides optional information about the agent's current state.	Blender Server
transfercount	The number of times the ADU was transferred since it was created. (historical)	ADU Server
ts.	Container names beginning with ts. are reserved for use by the Avaya Definity Telephony Server.	Definity Telephony Server
ts.<n>.loginid	The login id of the agent for the Avaya Definity switch.	Definity Telephony Server
ts.<n>.phone	Agent extension, where *n* is the contact number for the agent on the Avaya Definity switch.	Definity Telephony Server
ts.<n>.ptype	The Phone type–eas, acd.	Definity Telephony Server
		(Sheet 3 of 4)

Field Name	Description	Set By
ts.<n>.equip	Equipment number.	Definity Telephony Server
ts.<n>.starttime	Time (time_t) that the client assigned to the TS.	Definity Telephony Server
voice.acdname	The name of the ACD the TS is serving.	Telephony Server
voice.connector		
voice.connectorname		
		(Sheet 4 of 4)

Voice Data Containers

The following table lists the call containers in which end point events and attributes are stored. X represents the unique identification number for each end point. Y and Z represent sequence numbers within each end point's activities.

Note: The 6.0 style call containers are presented in this table. Call containers in the 5.5 style are still supported, and is documented in Appendix B of the *Telephony Services for the Avaya Definity G3, Release 5.5* manual.

Name	Value	Explanation
voice.X	delta time	For voice.1, this is base time (should be zero). For other cases, it is elapsed time in seconds since creation of the EDU.
voice.X.abandon	reason for abandoning the call: "in queue," "while ringing," "while on hold."	If the exit reason of a call end point is "abandon," this item provides additional details.
voice.X.agent_key	unique database record key	The key that retrieves the agent record from the database.
voice.X.conferencedest.Z	phone number	The destination phone number (Z) of a conference call, where Z = 1, 2, 3, ...
voice.X.connect	delta time	The elapsed time in seconds between the creation of the EDU and the time the call was connected.
voice.X.destination	phone number	Phone number of client at end point X.
voice.X.direction	inbound/outbound	The direction of the call to or from the agent.
voice.X.exit_reason	reason for exit: "normal," "transfer," "abandon," "other."	The switch supplies a reason for the termination of a call end point.
voice.X.holdtime.Y	time in seconds	Time spent on hold during hold instance Y.

Name	Value	Explanation
voice.X.leg_id	unique id (UUID)	Unique leg_id of the current leg of the call.
voice.X.loginid	loginid	Login id of client at end point X.
voice.X.origin	phone number	Phone number (ANI) at other end of end point X.
voice.X.queue	delta time	The elapsed time in seconds between the creation of an EDU and the time at which the switch reports the call as queued.
voice.X.queue_number	queue number	Queue number configured within the CallCenter switch as an Application Group.
voice.X.queuetime	time in seconds	Time reported by the switch between a call arriving on an inbound trunk of the switch and the call being sent to an agent queue. **Note:** Currently, this value is always zero (0). Subsequent releases of the telephony server will supply more meaningful values.
voice.X.ringtime	time in seconds	Time reported by the switch between agent phone starting to ring and agent answering the phone.
voice.X.talktime	time in seconds	Time reported by the switch between agent answering the phone and agent hanging up the phone.
voice.X.transfer	phone number	The phone number to which the call has been transferred.

Queue ADU Fields

The following ADU fields are used by queues on Avaya IC.

Field Name	Description	Set By
abandoned	The total number of contacts that have been abandoned before reaching an agent from this queue.	Media Connector Server
connector	The UUID of the media connector server responsible for the queue.	Media Connector Server
contactcount	The number of contacts currently in the queue.	Media Connector Server
		(Sheet 1 of 2)

Field Name	Description	Set By
contactsoffered	The total number of contacts received by the queue. The number of handled contacts is computed by IC Manager by taking the difference between contactsoffered and abandoned.	Media Connector Server
id	The media channel specific id of the queue. The id must be unique for a media channel in a specific site.	Media Connector Server
media	The type of media channel over which the queue is receiving contacts: voice, email, or web.	Media Connector Server
minimumagents	The fewest number of agents that should be assigned to one queue.	Media Connector Server
oldest	The arrival time of the contact that has been in the queue for the longest period.	Media Connector Server
priority	The priority level assigned to the queue. It can be 1 for the highest priority down to 10 for the lowest.	Media Connector Server
servicelevel	The interval within which a queued contact should be assigned to an agent on the system. This time period value varies across media channels.	Media Connector Server
servicelevelmiss	The number of contacts that were not handled during the prescribed service level time.	Media Connector Server
queue_key	The unique identifier for a queue, for reporting on queue activity. This is only set when a Queue is involved in the routing of the segment. This is not set when it is a direct transfer or route to an agent, or if it is routed through Advocate.	Media Connector Server
ttlqueuetime	The period of time that contacts have spent in the queue.	Media Connector Server
updatetime	The time of the last update to the queue.	Media Connector Server
youngest	The arrival time of the contact that has been in the queue for the shortest period.	Media Connector Server
		(Sheet 2 of 2)

Containers

A container is a grouping of values under a common name. Containers are trees of data within an ADU.

For example, within each ADU, the Telephony Server creates a container called ts, and within the ts container it creates a subcontainer for each of the agent's logical phones (ts.1, ts.2, and so on). Each data item pertaining to that logical phone takes a private subtree of the container–ts.1.phone, ts.1.ptype, and so on. Values for a second logical phone are written in fields ts.2.phone, ts.2.ptype, and do not interfere with values written for the first logical phone.

Note: For the Northern Telecom Meridian Link, the position ID and the IDN are considered separate logical phones, even though they share the same physical device.

The ts containers are maintained by Telephony within Avaya IC. You can create additional containers to suit your specific application.

Methods exist to return an entire container or subcontainer. Using the above example, to see all the data about the agent's first logical phone, you would use the GetSubTree() method on the ADUID and ask for "ts.1". With the exception of the GetSubTree() and DeleteSubTree() methods, there are no special container methods. Any method that involves getting or setting data can use container-based names.

Container Names and Special Tokens

You can specify container names or you can allow the ADU Server to generate container names for you based on your instructions. A number of special tokens are available for use when constructing container names. These tokens expand into appropriate strings and automate many of the steps involved in using containers.

Container names have multiple parts (two or more) separated by dots. Each part, called a name token, is limited to 1–35 alphanumeric characters (underscore is also permitted), with entirely numeric name tokens reserved for use with the special tokens described below. Up to 31 dots (32 tokens total) are permitted in a container name. Note that processor usage for container names is low, but is proportional to the number of tokens in the name.

There are four special tokens available for use with container names:

+ The simplest special token is **+**. This expands into a numeric token that is greater than the last numeric token created with a + in that position (for that container in that ADU). The first time it is used, it generates a 1. This means that if any client specifies a name of "code.+", the ADU server turns it into "code.1", but the next time any client uses "code.+" in that ADU, it expands into code.2. This makes it very easy to create subtrees within a container. The name generated in this way will never be the name of an existing subtree. (The gencount and padwidth configuration parameters, described in "Configuration Parameters," on page 37), can affect subtrees created with the **+** token.)

! The ! token expands into a numeric token that belongs to the client. Tokens that belong to a given client are created via a special use of the + token. Essentially, a subcontainer created with + is assumed to belong to a caller (usually the one who created it), and the ! token will find it.

To create a subcontainer for another client, add the loginid of the client after the + token that creates it, as follows:

“agent.+jane”

Whatever number it creates, it will be matched by a request for “agent.!” by client jane.

If several subcontainers have been created for the same user, the ! token seeks out the highest numbered (most recently created) one. If no subcontainer has been created for the client, the use of ! generates an exception.
To find another client’s subcontainer, add the loginid of the client after the ! token: “*containername*.!*loginid*”

null The null token contains no characters. It matches the subcontainer most recently created with the + token, regardless of who it was created for or who the caller is. This is most generally useful in creating and filling a subcontainer in one method call. For example, the following names given to SetValues:

“mydata.+”
“mydata..age“
“mydata..height”

would create a new, numbered subcontainer in mydata, and within that same subcontainer create age and height. If mydata.+ generated mydata.3, this would generate mydata.3.age and mydata.3.height.

Use of the null token should occur within the same method invocation as the use of the + token it relates to. Otherwise, some other application might use the + token within the same container and ADU, and change the meaning of the null token unexpectedly. However, if the intent is to add data to the most recently created subcontainer no matter who created it, the null token is appropriate.

* The * is permitted only within Assign methods and is described in “Monitoring Criteria: Wildcards,” on page 32.

Limitations of Container Syntax

The first token in a container name cannot be a special token.

When a * token has been used, special tokens other than * cannot be subsequently used.

Note that some methods do not permit some tokens. Most do not support * and some (GetValues and methods like it) do not (meaningfully) support +. For example, in the method invocation GetValues(“a.+.x”), the + token would be interpreted as “create the next number in the sequence,” which is not meaningful for the GetValues method, as the name being generated cannot exist yet and so cannot be read..

Container Configurations

The following TS configuration parameters were agreed upon with regards to containers:

Parameter	Type	Default	Description
aducon	bool	false	Turns ADU Containers on/off
tscon	bool	false	Turns EDU Containers on/off
containers_56_style	bool	false	Determines if “state” statistics in the old style (5.6) are written.
containers_60_style	bool	true	Determines if “state” statistics in the new style (6.0) are written.

Following are examples of the results for a call scenario where an agent receives a queue call, hold and retrieves it, blind transfer to a second agent who concludes the call.

Values ALWAYS written:

ADU.Create		VDU.Create	
	loginid		ani
			dnis
			primary_ani
			primary_dnis
			loginid
			agent_key
ADU.Update		**VDU.Update**	
	voice.contactcount		phone
	voice.acdname		ani
	voice.connector		dnis
	voice.connectorname		dest
	voice.device		orig
	voice.eduid		ext
	voice.contactsoffered		loginid
			agent_key
	voice.state.loggedin		agent.+
	voice.state.available		
	voice.state.active		
	voice.state.loggedout		

EDU Values written if tscon is set to true:

voice.1.loginid
voice.1.leg_id
voice.1.agent_key
voice.1.destination
voice.1.origin
voice.1.direction
voice.1.connect
voice.1.holdtime
voice.2.loginid
voice.2.leg_id
voice.2.agent_key
voice.2.destination
voice.2.origin
voice.2.direction
voice.2.connect
voice.1.exit_reason
voice.2.transfer
voice.2.ringtime
voice.2.queuetime
voice.2.talktime
voice.2.exit_reason

ADU and EDU values written if containers_56_style is set to true:

voice.1.state	(value="alerting")
voice.1.state.alerting.starttime	
voice.1.state	(value ="incall”)
voice.1.state.incall.starttime	
voice.state.incall.total	
voice.1.state	(value="hold")
voice.1.state.hold.starttime	
voice.state.hold.total	
voice.2.state	(value="incall")
voice.2.state.incall.starttime	
voice.1.state	(value="disconnected")
voice.1.state.disconnected.starttime	

ADU and EDU Values written if containers_60_style is set to true:

voice.1.stdstate.1009981854	(value="alerting”)
voice.1.stdstate.1009981854.alerting.reason	
voice.1.stdstate.1009981856	(value="active")
voice.1.stdstate.1009981856.active.reason	
voice.1.stdstate.1009981858	(value="inactive")
voice.1.stdstate.1009981858.inactive.reason	
voice.2.stdstate.1009981863	(value="active")
voice.2.stdstate.1009981863.active.reason	
voice.1.stdstate.1009981863	(value="terminated")
voice.1.stdstate.1009981863.terminated.reason	(value="101")

CHAPTER 3
EVENT MONITORING

ADU Event Monitoring

This chapter describes the events that are sent by the ADU Server. This chapter also explains how to assign a request on behalf of a client and how to establish monitoring criteria. The ADU Server continuously monitors the contents of ADUs for changes, and reports those changes to interested clients through event messages.

Clients register their interest in ADUs by assigning to the ADU Server with a monitoring criteria, which is a specification describing the condition that should generate an event notice. For example, a client might request notification when an agent enters wrapup state, or when an ADU is terminated. The client receives event messages about ADUs that match the specification, and continues to receive all events generated for those ADUs for as long as they match the client's monitoring criteria.

The ADU Server generates seven types of events that describe changes to existing ADUs. The following table lists the events and the contents of the event messages written in the ADU and sent to monitoring clients.

Event	**Description**	**Message**
ADU.change	A modification has occurred in a monitored ADU.	All affected {name, value} couples and the ADUID.
ADU.delete	Values have been deleted from the ADU.	All deleted values.
ADU.drop	The ADU has changed and no longer matches the assign criteria. It will no longer be monitored. If the ADU later changes so that it again meets the assign criteria, the client gets an ADU.watch event.	The ADUID.
		(Sheet 1 of 2)

Event	Description	Message
ADU.end	The ADU has been terminated. The server passes an ADU.end message to all monitoring applications.	All of the data elements in the ADU.
ADU.transfer	The ADU has been transferred.	The new client and the ADUID. (Note that ADUs are not generally transferred. This event is relatively rare.)
ADU.watch	The ADU matches the monitoring criteria.	All of the data elements in the ADU.
		(Sheet 2 of 2)

When a client first assigns monitoring criteria to the server:

- All existing ADUs are checked to see if any match the specification. All subsequently created ADUs are checked for a match.
- Any ADUs that do match the criteria are placed under watch and an ADU.Watch event message is sent to the monitoring application.
- Changes to ADUs under watch are reported to the application by an ADU.Change event.

Note: When a client has assigned to the ADU Server, it should monitor the event data stream for the data it wants, rather than invoking ADU.Get…() methods repeatedly.

As values are set in the ADU, the ADU Server sends Change events to assigned clients. The timing, order, and content of the Change events depend on the components setting the values, are is therefore variable.

Starting and Stopping Event Monitoring

The ADU Server methods ADU.Assign() and ADU.Monitor() let you set up monitoring conditions. The method ADU.Deassign() lets you revoke them.

By default, a client does not receive change events for the ADU changes that it itself initiates. This cuts down on needless event traffic. It does receive other sorts of events, as well as event messages for changes caused by other clients. If the change causes a .Watch or .Drop event, it is reported to the client. To receive change events for the client's own changes, prefix the Assign or Monitor expression with a plus symbol (+):

For example:

```
[ADU.Assign("+*")]
```

(The asterisk (*) in the example indicates that all ADUs in the local ADU Server should be monitored.)

Assigning to the ADU Server and monitoring an ADU do not add a client's name to the internal list of ADU-modifying clients. (The internal list of clients is described in “ADU Termination,” on page 12.) Assigning allows the client to watch the activity of an ADU. The client does not have to issue an ADU.Terminate() for each ADU being watched.

By default, a client's assign criteria is distributed to all ADU servers. If this is not desired, prefix the Assign or Monitor expression with a colon symbol (:):

For example:

```
[ADU.Assign(":loginid=wynn")]
```

If you specify both : and +, : must come first. So ":a=b" watches field a for value b, and reports only on those local vdus.

Note: In this sense, local means "this VDU server". If you fail over to a different VDU server, you may not get the result you were expecting.

iver=148

To test, try: [ADU.Assign(":state#Z")]

In a WAN, your ADU server should show events coming back for all logged in agents, but the other ADU servers should only get RemoteWatcher requests with empty strings, and should send no events back.

Setting Event Monitoring Criteria

ADUs are selected for monitoring by criteria matching. A **criteria** is an expression that selects some ADUs and rejects others according to a defined specification. If a criteria expression is provided as a parameter to the Assign() or Monitor() method, each name/value pair in the ADU is tested against it. If the criteria determine that a match exists, the ADU is watched. Subsequent changes are sent to the client as change events.

The following values cannot be successfully used in an Assign criteria:

- createtime
- createtimet
- duration
- endtime
- suspend
- termination
- transfercount.

Monitoring Criteria: Syntax

The general syntax of a criteria statement is:

```
method  (name relationalop value [booleanop] name relationalop value)
```

```
Example:   ADU.Assign "loginid=Joe & ts.1.s=wrapup"
```

In the above example, the client has assigned to the ADU Server, asking it to watch for any ADU that contains the following two values:

- "Joe" in the field **loginid**
- "wrapup" in the ts.1.s field.

In other words, notify the client every time Joe is in the wrapup state.

An example of a selection criteria string when making an ADU monitor request:

```
//[ADU.Monitor("adu_id=387cdde0000100000a6403b81f450002 |( type=per &
name=jonathan & connectrate=* ) | ( type=queue & name=phone_queue & idletime=*
)")]
```

Note the spaces between the parentheses and the type designation.

You can construct complex statements that evaluate multiple conditions by grouping expressions together with parentheses to control the order of evaluation.

A single * character has a special meaning when used as a monitor criteria. It causes all local ADUs (those in the local ADU Server) to be watched. This is used by the IC Manager and should generally be avoided by other applications, as it generates a very large amount of event traffic.

Setting the monitor criteria to a null value ("") stops monitoring.

The Assign method does not accept special tokens !, +, or the *null* token in container names. These tokens have specific meanings for a particular ADU and vary from ADU to ADU. The Assign method has no provision for names that do not have ADU-independent meaning.

The Assign and Monitor methods contain criteria strings that allow you to use additional syntax. At the end of the string is a list of fields, called a projection list, enclosed in curly brackets where you can specify filter criteria. The list is comma separated with no spaces between the fields.

For example, `{specification,specification...}`

You can specify the type of field name in three ways:

1. A simple fieldname, such as a, b, or c.
2. A wildcard container name, such as a.* or a.*.d. The wildcard designation (*) can be used after a dot (.), at either end of the specification, or followed by a dot (.) in the specification.
3. A request to retrieve all of the fields below a specified point in a container tree, as in "a.b?". This format collects all of the field names below the specified point in container tree. It does not collect field names at its own level or higher on the tree.

Example: `+vdu_id> "" {call?,data.*,loginid}`

Note: Spaces are optional after the selection criteria and before the projection list. The projection list itself is optional, but no filtering is done without one. If a projection list is used, it cannot be empty.

- This assign criteria watches all EDUs in the system because all EDU IDs are longer than empty strings.
- Change events that do not reflect a change in the loginid, any matching **data.***, or any subcontainer of the call container are also suppressed.
- Change events that not suppressed are trimmed to list only the names that are provided on the projection list. The duid, time, and modifier fields are also provided.

Note: Watch and drop events are not modified and they are always sent with their full content. User events, which are very rare, would be modified as they are actually change events with a different name.

Relational Operators

Values can be compared to each other as described below. There are four permissible relational operators. In the description column, the term *couple* refers to a name/value pair.

Symbol	Definition	Description
=	equal	Find a couple that matches the specified name and value. Wildcards are accepted.
<	less than	For a given name, find all couples with values less than the specified search value (string-based).
>	greater than	For a given name, find the set of couples with values greater than the specified criteria (string-based).
:	exactly equal to	For a given name, find a couple that exactly matches the specified value. Wildcards are not accepted.
#	not	Expressions are not equal to each other.

The ADU Server compares the name/value data within an ADU against the criteria string one character at a time, starting with the leftmost character, until either a mismatch occurs or one field runs out of characters. Comparisons are case-sensitive. “Foo” is not the same as “foo” or “FOO”. To avoid ambiguity, enclose the specified value within double quotes (“”). Within a quoted string, (\") means quote and (\\) means backslash.

If the strings match character for character, they are equal. Strings can also be relatively compared. If one field runs out of characters without a mismatch, the relatively longer field is greater (for example, abc < abcd). If a mismatch occurs, the field containing the greater character is greater, (for example, 111 < 119). Because the comparison is string-based, not numeric, the field with the greater initial character is greater, (for example, 2 > 119).

Wildcards are permitted only with the equal (=) operator.

Boolean Operators

Boolean comparisons that return evaluations of true or false can be performed between two values. The two boolean operators are described below.

Symbol	Definition	Description
&	and	Both expressions must be true.
|	or	True if one or both expressions are true.

For example:

```
ts.1.s=Available | ts.1.s=WrapUp
```

selects all ADUs in which the agent state is either Available or WrapUp. Terms may also be grouped with parentheses for more complex expressions.

Monitoring Criteria: Wildcards

Instead of specifying each individual instance of a set, you can use wildcard symbols to stand in for other values. The wildcard symbols can be used in setting criteria given the following restrictions:

- Wildcard usage is restricted to the equal (=) relational operator. You cannot use wildcards with the less than (<), greater than (>), or exactly equal to (:) operators.
- Single character wildcards must have a character to match.

Each (?) symbol can stand in for one character. The (*) symbol can either be placed at the end of a value (a so-called value trailing wildcard), or can be entered by itself to find ADUs with an instance of any value (selects all ADUs as matching).

Wildcard	Definition	Example
?	Match one character	ts.1.ptype = ??? Find all couples named “ts.1.ptype” which contain a three-character value.
*	Match an unlimited number of characters.	ts.1.s.logout = * Find all ADUs for agents that have logged out of the teleset.

Container names used in an Assign can use limited wildcarding. Within an Assign, an expression like

```
ts.*.s = Available
```

selects any ADU in *any* subcontainer in ts in which the state is Available. Note that the **!** token is not available in an Assign, but this use of the wildcard token can be used instead.

Monitoring Criteria: Examples

The following examples demonstrate how to instruct the server to monitor ADUs that fulfill specific criteria. As you can see, there is flexibility in specifying monitoring criteria. Choose the method that best fits the current circumstances.

Criteria Example	Description
ts.*.phone = "1234" or ts.*.phone : "1234"	Monitor all calls for agent extension 1234.
ts.*.phone > "0999" & ts.*.phone < "2000"	Monitor all calls for agent extensions 1000 through 1999.
ts.*.phone = "1???"	Monitor all calls for agent extensions 1000 through 1999.

CHAPTER 4
ALARMS

IC Manager provides system administration tools for monitoring alarm events. Visual and sometimes auditory alarms (beeps) are triggered whenever the system detects problems that require human intervention, such as server failures. Alarms are categorized as Emergency, High, Low, or Informational. (For more information about monitoring alarms, refer to *IC Administration Volume 1: Servers & Domains*.)

To minimize Alarm Server traffic, alarms that occur repeatedly are gathered by the ADU Server and raised periodically. The repeat count indicates how many times the condition has occurred since the alarm was last raised.

The following table describes the alarms that are associated with the ADU Server.

Alarm Name	**Priority**	**Description**	**Cause/Recommended Action**
AssignFail	Emergency	Cannot Assign to ADU%s, ignoring it	Another ADU Server across the WAN is unavailable. A site has been cut off. WAN features may not be available.
BadADUInDS	Emergency	Bad UUID %s in DS's list of ADU servers	Corruption in the DS has made it impossible to identify other ADU Servers for WAN work. Contact Technical Support (1-800-886-3244).
DumpADU	High	ADU overflow, killing eldest	The ADU limit set by the adus parameter has been reached. The oldest ADUs are being removed to make room for newer ones. Increase the value of the adus parameter. Determine if ADUs are being left in the server when they are no longer needed, and possibly adjust timer settings (nouserinterval, idletime).
			(Sheet 1 of 2)

Alarm Name	**Priority**	**Description**	**Cause/Recommended Action**
FailADUCon	High	Connection to *<uuid>* closed; *n* dropped watchers *[reason]*	A connection to a remote server has failed. The UUID of the remote server is reported, as is the number of clients that were assigned to the server. The reason is often ADU.ServerFailed.
LostADUEvents	High	*n* events lost due to network congestion or error	Event handling requests are timing out. *n* represents the approximate number of events that have been lost.
NoMeInDS	Emergency	This ADU Server is not listed in the domain's DS for this ADU Server.	The configuration of servers and domains is incorrect. Contact Technical Support (1-800-886-3244).
NoEventSink	High	A VESP_Request call failed to send a request to the server named by the EventSink configuration parameter.	The server could not be reached. History is being lost. If you are not using a server to archive terminated ADUs, set the database configuration parameter to 0 and restart the ADU Server. Check the vesp.imp and vespidl.pk installation files. If they do not list the server and interface, re-install the files and restart the ADU Server.
Victims	High	Shutdown with *n* ADUs in existence; objects discarded	This alarm should not be generated. Contact Technical Support (1-800-886-3244.)
			(Sheet 2 of 2)

CHAPTER 5
ADU SERVER CONFIGURATION

System Considerations

The Max Active Adus configuration parameter, described below, should be set with consideration for system capability. The number of ADUs that can be effectively handled by the ADU Server is proportional to the system's available memory and processor speed. A typical ADU requires 40K in memory. Active agent sessions might require 60K or more. Memory can be conserved by using the same data names in many ADUs. The memory space for name storage is shared. The memory used for storage of values, however, is not affected.

Processor usage may be decreased by setting values in groups as opposed to one at a time. When possible, gather sets of name/value pairs and apply them all at the same time.

ADU Server Alias Name

When setting an alias name for the ADU Server, note that the name “localADU” is reserved. In a WAN environment, it is used to segregate an ADU Server that has been taken offline. Refer to *IC Administration Volume 1: Servers & Domains* for information on setting alias names for servers.

Configuration Parameters

The ADU Server is configured through the IC Manager. Refer to *IC Administration Volume 1: Servers & Domains* for configuration instructions. The following table lists the label used when configuring with IC Manager, followed in parentheses by the parameter name as it is required internally by the server.

Label	**Description**
Checkpoint frequency (checkpoint.interval)	Minimum interval period in seconds between requests to checkpoint a specified ADU into the DUStore server. A value of –1 means do not checkpoint.
Idle Time (idletime)	Number of minutes an ADU may remain inactive before being terminated. The default is 90 minutes. Minimum is 1 minute, maximum is 30 days.
	(Sheet 1 of 4)

Label	Description
No User Interval (nouserinterval)	Minimum number of seconds an ADU may linger in memory when there are no users active for it. Default is 60 seconds. Minimum is 1 second, maximum is 90 minutes.
Random Kill Interval (randomkillinterval)	Maximum number of seconds an ADU stays in memory after the usual timers have expired. The default is 30, the range is 1 to 180 seconds. Random intervals are useful when a large number of ADUs are simultaneously suspended or deleted, causing the DUStore to be flooded with requests. Handling the requests over a 2-minute period decreases server stress. Where such a situation is unlikely, more predictable timing and better memory usage result from a setting of 1. While testing to see if ADUs are being retired when they should be, 1 is also an appropriate setting.
Scan Interval (scaninterval)	Number of seconds to wait between checking various ADU Server timers. Higher values may save some CPU time. Lower values make for more predictable behavior during prototyping and testing. Default is 30 seconds. Minimum is 1 second, maximum is 60 seconds. Assume that other timers in the ADU could be off by as much as (this interval + 1) to start.
Max Active Adus (adus)	The maximum number of ADUs that the ADU Server keeps active at the same time. If more than this are created, it sends an alarm and forcibly terminate the oldest one to make room for each new one. This value should be somewhat greater than the number of agents using this server to handle calls. The default varies by release. **Always set this value explicitly.** The number of ADUs that can be effectively handled by the ADU Server is proportional to the system's available memory and processor speed. A typical ADU requires 40K in memory. Active agent sessions might require 60K or more.
Watchers (watchers)	The number of clients who are interested in assigning to ADU Servers. To start, this should be equal to the number of agents in the contact center (across all contact centers in a WAN environment), plus a few extra. **Default varies by release. Always set this value explicitly**, as the default is very large and some memory is wasted in each ADU if the value is set too high. If set too low, Assigns are rejected.
Database (database)	Should be checked if you want to enable the use of HISTADD reporting, unchecked if not. The default is unchecked.
Pool Size (poolsize)	The initial amount of memory allocated for data belonging to each ADU. Increase the pool size if a large amount of data is stored and performance needs to be improved. The default is 2048.
Pool Growth Increments (poolgrowsize)	The amount of memory to add to the string pool for each ADU if the pool runs out. Increase the pool size allocation when more memory is needed to store strings or events. The default is 1024.
Number of fields (initialdatalength)	The number of fields expected in an ADU. This does not limit the number of fields, but when this number is exceeded, the server must reallocate space. The default is 128. If using containers, this value should possibly be increased to 256 or 512.
Pool Re-Pack (%) (repackfree)	When the specified percentage of the pool belonging to an ADU pool becomes free, the ADU Server repacks the pool to save memory. Sites at which performance is critical and memory is plentiful may consider using a value of 100. The default is 25.
	(Sheet 2 of 4)

Label	Description
DUStore (dustore)	Enables the use of the DUStore Server. Check the check box to enable or uncheck to disable.
DUStore ADU Batch Size (maxkills)	The maximum number of ADUs that are sent to the DUStore Server in one set. The minimum is 1 and the default is 60.
Poll Wait (pollwait)	Sets the interval for the Toolkit's Select() method, in hundredths of a second. The default value is 5 (50 milliseconds). Low values increase CPU utilization. High values (over 100) may affect the accuracy of the scaninterval parameter.
Maximum number of Revisions (maxkeptrevisions)	Determines the number of revisions of a value that are kept for access with the GetValuesHistory() method. The default is 4, which is generally recommended.
Number of Cached Adu events (keepevents)	Determines the maximum number of events to be kept in memory for each ADU. The default is 256. A setting of 64 is recommended as a reasonable number of events to be kept in memory. High values may increase Eventsink throughput. Low values may conserve memory.
Retry Interval (serverretryinterval)	Specifies the number of seconds to wait between automatic attempts to reassign to servers. The minimum is 6 seconds, maximum is 48 hours, default is 1 minute. Setting this value below 30 seconds in a production environment is not recommended, as it does a synchronous DS call for a list of ADU Servers and attempts an asynchronous Assign to any of them it isn't already assigned to. If the Assign fails (the request function itself fails), the offending ADU Server is removed from the list and not retried automatically. The retry is only attempted if the Assign callback reveals an error or a ServerFailed event arrives.
Reset Interval (serverresetinterval)	Specifies the number of seconds to wait between attempts to reassign to servers after receiving an ADU.FailADUCon alarm. The minimum is 0, maximum is 48 hours, default is 2 seconds.
Data Element Names	
Suspend Interval (suspendinterval)	Number of seconds before an ADU, which is suspended by use of the Suspend method by all users, is considered for Suspension. This parameter can be overridden by a higher value in the Suspend method. The minimum is 1, maximum is 20 minutes, default is 1 second.
Adu Data Percent (adudata.perecnt)	The percentage of ADUs that are sent to the ADU data feed, used for random sampling. The minimum is 0, maximum is 100, default is 0.
Find Create to Search (findcreatestoresearch)	Enables a WAN wide DUStore search on the FindOrCreate method. Check the check box to enable or uncheck to disable.
DUindex Lookup1 (duindex.lookup1)	The name of one of the fields used to index the ADU in the DUStore Server. Used with the Find method.
DUindex Lookup2 (duindex.lookup2)	The name of one of the fields used to index the ADU in the DUStore Server. Used with the Find method.
DUindex Info1 (duindex.info1)	The name of one of the fields used to identify the ADU in the DUStore Server. Used with the Find method.
	(Sheet 3 of 4)

Label	Description
DUindex Info2 (duindex.info2)	The name of one of the fields used to identify the ADU in the DUStore Server. Used with the Find method. The default value is media.
DUindex Info3 (duindex.info3)	The name of one of the fields used to identify the ADU in the DUStore Server. Used with the Find method. The default value is site.
Adudata Alarm Priority (adudata.alarm.priority)	An alarm priority used when raising an alarm related to the ADUDATA feed. Values are Low, High, Informational, and Emergency.
Adudata Event Name (adudata.eventname)	The names of events that may be sent to the ADUDATA feed. If there are no events identified, all events are sent. Use this parameter multiple times to specify multiple events.
Adudata Event Ifname (adudata.event.ifname)	Sends an event to the ADUDATA feed only if the event contains this field. This parameter can be specified multiple times to allow multiple valid names.
Adudata Data Only Name (adudata.data.onlyname)	Filters events to the ADUDATA feed so they only contain this field. The parameter can be specified multiple times to allow multiple valid names.
Server Group servergroup	Reserved
Server Status serverstatus	Reserved
	(Sheet 4 of 4)

The following configuration parameters are not presented on the ADU tab in IC Manager. You can set them on the **Config** tab of the ADU Server Editor dialog.

Name	Description
filter (filter)	Sets the filter for determining which ADU events are sent to the HISTMAP server (the Report Server). The event types are start, change, delete, transfer, user, and data. A plus sign (+) marks the event type for storage, a minus sign (–) excludes the event type from storage. *End* events cannot be filtered out. The *data* parameter encompasses all event types. *-data* is the default. Each filter criterion must be entered on a separate line, with subsequent lines taking precedence. Examples: *–data* *+change* only *change* and *end* events are stored *+change* *–data* only *end* events are stored Note that *drop* and *watch* events are sent to clients but are not included with the *+data* filter. aduhskeepname (reportkeepname) Names of data elements in ADU events to be sent to the server specified with the Eventsink configuration parameter. Each element must be entered on a separate line. If this parameter is not used, all data elements are sent. (The ADUID field is always stored.) Use of wildcards is permitted. Use this parameter with care. Filtering elements from the End event may adversely affect reporting capability.
gencount	Specifies the number of instances of a subcontainer created with the + token that can exist at one time. If more than this number are created the earliest is deleted. For example, to limit the number of subcontainers of the ts container, you could set `gencount` to ts.3. When ts.+ creates ts.4, ts.1 is deleted. When ts.5 is created, ts.2 is deleted, and so on. The default is 0, which specifies no limit (all generations of a subcontainer are kept). The gencount value should be set to accommodate the maximum number of active contacts for each media channel.
padwidth	Specifies the number of digit positions required in a container name token. Numbers are padded with zeroes to this position. For example, to specify the number of digit positions for the second token of the ts container, you could set `padwidth` to ts.4. Subcontainers would then be named ts.0001, ts.0002, and so on. The default is 0, which specifies no padding.
autostart	Specifies whether the server should start automatically at the same time as the ORB Server. Values: true, false.
timetype	Sets ADU server timestamp format. The only supported value for this version is gmt (lower case).

The following settings are on the Debug tab of the ADU Server Editor.

Name	Description
logfilesize	The maximum size of the log file. The default log file size is 2,560,000 bytes. The minimum is 1000 bytes and the maximum is 50,000,000 bytes.
Ping Interval pingtimer	The number of seconds between messages that are sent to a server to determine if it is running. The default is 60 seconds and the minimum is 1 second.
trace	The trace parameter can contain any number of the following settings, comma separated, no spaces permitted. This following list works for MTT servers. STT servers have some minor differences.. idl – Whether requests, responses and events are logged flush – Whether the logged information if forced to disk after each significant operation timing – Whether requests have timing information logged explain – Whether requests explain which server they were sent to, and why on – Whether logging happens at all thread – Whether each log entry contains the id of the thread that wrote it usr1 through usr8 – For use by third party implementers. dev1 – reserved appverbose – Used by individual applications. appwarn – Used by individual applications. debug – Used by individual applications. debug2 – Used by individual applications.

Note: Previous versions of Avaya IC included the ADU History Server. To support existing installations, send the events to an existing ADU History Server.

1. Use the default for the Eventsink parameter (HISTMAP.EventsIn) to send events to the Reporting server.
2. Set the Reporting servers's eventforward parameter to ADUHS.EventsIn. The Reporting server forwards all ADU events to the ADUHS.

CHAPTER 6
IDL SPECIFICATION

The Interface Definition Language (IDL) is defined within CORBA standards. It is used to create interfaces that are called by client objects and provided by object implementations.

The following is the IDL description of the ADU Server. The virtual interface is inherited by both the EDU Server and the ADU Server. Several of the methods listed are specific to the ADU Server, and are therefore omitted from the method descriptions in Chapter 7.

```
interface genericdu virtual {
ORBStatus Create( in SeqCouple values, out stringvduid );
ORBStatus Terminate( in stringvduid );
void TerminateMine();
ONEWAY EventsIn(in string vdu_id, in SeqEvent events);
ORBStatus FindByKey( in string name, in string value, out stringvduid );
ORBStatus Find( in string search_criteria, in unsigned long, scope,
out SeqString matches );
ORBStatus FindExtended( in string search_criteria,
in unsigned long scope, out SeqCouple matches );
ORBStatus GetOneValue( in string vduid, in string name,
out string value );
ORBStatus GetValues( in string vduid, out SeqCouple c );
ORBStatus GetActive( out SeqString vduseq );
ORBStatus SetOneValue( on string vduid, in string name, in string value );
ORBStatus SetValues( instring vduid, in SeqCouple values );
ORBStatus DeleteValue( in string vduid, in string pattern );
ORBStatus IncrValue( in string vduid, in string name, in long incr,
out string newvalue );
ORBStatus Assign( in stringmonitorcriteria );
ORBStatus Monitor( in stringmonitorcriteria );
ORBStatus Transfer( in string vduid, in string to );
ORBStatus SetAndTransfer(in string vduid, in string to,
in SeqCouple values );
ORBStatus GetValuesHistory( in string vduid, in unsigned long flags,
out SeqLong headers, out SeqString names,
out SeqSeqSeqString values );
ORBStatus GetValueHistory( in string vduid, in string name,
out SeqString values, out SeqString when,
out SeqString who );
ORBStatus GetSubTree( in string vduid, in string name,
out SeqCouple matches );
ORBStatus GetSomeValues( in string vdu_id, in string name,
out SeqCouple matches );
```

```
ORBStatus SetValuesExtended( in string vdu_id, in SeqCouple data,
out SeqString newnames );
ORBStatus DeleteValues( in string vdu_id, in SeqString names );
ORBStatus DeleteOneValue( in string vdu_id, in string name );
ORBStatus DeleteSubTree( in string vdu_id, in string name );
ORBStatus SetAndTerminate( in string vdu_id, in SeqCouple data );
void GetUserSessions( in string vdu_id, out SeqString users );
//Server ONLY
ORBStatus RemoteWatcher( in unsigned long handles, in string cri );
ORBStatus ForwardEvent( in unsigned long handles, in Event Event );

void SetDefaultHistoryFilter( in unsigned long mask );
ORBStatus SetHistoryFilter( in unsigned long mask );
}

interface ADU : genericdu, General {
const unsigned long NO_OS = 0x00;
const unsigned long ASK_OS = 0x01;
const unsigned long NET_NO_OS = 0x02;
const unsigned long NET_ASK_OS = 0x03;

};
```

CHAPTER 7
ADU SERVER METHODS

Method Objectives

Clients request Avaya IC servers to perform various functions by issuing server-specific method invocations. These methods behave in a similar fashion across all servers. For example, when you invoke any of the various ADU Server Set methods, existing values are overwritten. Values that did not previously exist are created. With the exception of the Delete methods, all of the ADU Server methods generate change events to indicate changes or additions. If an error occurs, exceptions are raised.

This chapter defines the methods provided with the ADU Server.

Note: Some program examples were formatted to fit the page. Actual program lines cannot be arbitrarily split.

Exception Information

The ADU Server methods may return the following exception information:

Cannot use empty token
Improper special token for this method
Improper use of a special token
No listing for home server for that ADU
No such name: <name>
No such ADU: <aduid>
No WAN ADU services in this version
Not an ADU id: <aduid>
Server only attribute
Too many characters in token
Too many dot separators
Too many names at the same time
Unknown error
ADU is read-only

You are not a server
Cannot find subcontainer for that owner
First token must be normal here
No match (or illegal usage)
Cannot access invalidated member

Routing Requests

In an environment with several ADU Servers, any method that accepts an ADUID routes the request to another ADU Server if the ADU named is not local to the first server. ADUIDs contain location data, so a server can identify which system owns each ADU.

Method Overview

The following is a brief description of the methods available for use with the ADU Server.

ADU.Assign	Create a session with the ADU Server.
ADU.Create	Create a new ADU.
ADU.Deassign	Destroy a session with an ADU Server.
ADU.DeleteOneValue	This is equivalent to DeleteValues with a single name.
ADU.DeleteSubTree	DeleteOneValue on the given name and every name in the subcontainer beneath it, cutting a branch from the tree a container represents.
ADU.DeleteValues	Given a list of names, this method removes from the ADU the name and its value history, in effect making the ADU smaller.
ADU.EventsIn	Adds user-defined ADU events to an ADU.
ADU.Find	Returns a list of ADUIDs that match a simple criterion.
ADU.FindByKey	Finds one active local ADU based on a single search criterion.
ADU.FindOrCreate	Searches for a specified ADU and, if not found, creates the ADU.
ADU.ForceTerminate	Forces theADU object to be deleted from memory and from the database both, so it can no longer be used.
ADU.ForwardEvent	Reserved.
ADU.GetActive	Finds all the active ADUs at the time the call is made.
ADU.GetOneValue	Retrieves one value from an ADU.
ADU.GetSomeValues	Returns all names and values matching a container name.
ADU.GetSubTree	Returns all names and values in the subcontainer named by *name*.

ADU.GetUserSessions	Returns the sessions of all clients believed to have an interest in the ADU.
ADU.GetValues	Retrieves all of the values of an ADU.
ADU.GetValueHistory	Returns everything that is known about the named field's values in an ADU.
ADU.GetValuesHistory	Returns everything that is known about all values in an ADU.
ADU.IncrValue	A useful alternative to using SetOneValue and GetOneValue to modify a value when there is a risk that two applications might conflict, and the value being changed is numeric.
ADU.Monitor	Changes Assign criteria.
ADU.RemoteWatcher	Reserved.
ADU.SetAndTerminate	Combines a SetValues and a Terminate.
ADU.SetAndTransfer	Combines SetValues and Transfer into a single call, as these operations often occur together.
ADU.SetDefaultHistoryFilter	Reserved.
ADU.SetHistoryFilter	Allows the caller to specify which types of events are saved when an individual ADU is sent to the eventsink server, overriding the default history filter for one ADU.
ADU.SetOneValue	Sets one ADU data element.
ADU.SetValues	Sets one or more ADU data elements.
ADU.SetValuesExtended	Performs a SetValues, but in addition returns a parallel list of names that it created while resolving container names.
ADU.Terminate	Removes the client's name from the list of interested parties for one ADU.
ADU.TerminateMine	Removes the client's name from the list of interested parties for all ADUs.
ADU.Touch	Accesses or “touches” an ADU onceper invocation, as a default, to prevent it from being harvested from memory out of idleness.
ADU.Transfer	Adds a new client to an ADU's list of interested parties.

Methods

The following sections describe the ADU Server methods.

ADU.Assign

IDL Syntax `ORBStatus Assign( in string monitorcriteria ) ;`

Description Create a session with the ADU Server. When a session is created, events are sent to the assigned Avaya IC client.

When multiple ADU Servers are in use, **Assigns** watch all calls in the domain of the ADU Servers and notify the client with events when they occur. This makes Assign a relatively expensive operation. Design clients so they only need to Assign once to specify the ADUs in which they are interested. Assigning to multiple ADU Servers is unnecessary and causes numerous problems. (Refer to "Cooperation of ADU Servers," on page 10 for additional information.)

Input Parameters

monitorcriteria	Information used to select ADUs for monitoring. If values contain anything other than letters and numbers (for example, spaces), they should be enclosed in double quotes, and \ or " characters must be quoted by a \ character. (Refer to "Starting and Stopping Event Monitoring," on page 28 and "Setting Event Monitoring Criteria," on page 29.)

Returns

VESP_SUCCESS	Request was successful..
VESP_ERROR	Request was unsuccessful.

C Program Example

```
status = Vesp_Assign_Request( "ADU.Assign", callback, user_data,
                              event_callback, session, "loginid=user" );
```

ADU.Create

IDL Syntax `ORBStatus Create( in SeqCouple values, out ADU_ID aduid ) ;`

Description This method creates a new ADU. This function is usually performed by the Toolkit and is hidden from normal application development. The ADU Server sets the creation timestamp and ADUID.

Input Parameters

values	Initial values of the ADU, not to exceed 1023 values.

Output Parameters

aduid	Agent Data Unit Identifier.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	Internal error in ADU Server.

C Program Example

```
_IDL_SEQUENCE_Couple *seq_couple;
ADU_ID aduid; /* receives the id of the created ADU */
/* Create space for values */
seq_couple = vesp_couple_seq_create();
/* fill in the field for the new ADU entry */
vesp_couple_seq_add_couple_values( seq_couple,
                                   "name1", "value1" );
vesp_couple_seq_add_couple_values( seq_couple,
                                   "name2", "value2" );
vesp_couple_seq_add_couple_values( seq_couple,
                                   "name3", "value3" );
/* Create the new ADU */
status = Vesp_Request( "ADU.Create", callback, 0x2132, session,
                                   seq_couple, &aduid );
vesp_couple_seq_delete( seq_couple );
```

ADU.Deassign

IDL Syntax `ORBStatus Deassign( ) ;`

Description Destroy a session with an ADU Server. When a session is deassigned, the flow of events from the ADU Server to the client ceases. Events may continue to arrive until the response for this method is received. It is not an error to deassign when no session exists.

Returns

VESP_SUCCESS	Request was successful.

ADU.DeleteOneValue

IDL Syntax `ORBStatus DeleteOneValue( in ADU_ID aduid, in string name) ;`

Description This is equivalent to DeleteValues with a single name.

If used with a container name (“a.b”), you only delete that one name, a.b. However, the ADU Server methods are not able to address names “below” that point. Although a.b.c may still exist, the ADU Server cannot find it, even though GetSubTree on “a” still sees them. Resolving a.b.c depends on resolving a.b, which does not exist anymore. Callers should generally *not* delete elements from non-leaf positions.

Input Parameters

aduid	Agent Data Unit Identifier.
name	Name whose values are to be deleted from the ADU.

ADU.DeleteSubTree

IDL Syntax `ORBStatus DeleteSubTree( in ADU_ID aduid, in string name) ;`

Description This method does a DeleteOneValue on the given name and every name in the subcontainer beneath it, cutting a branch from the tree a container represents. If given a simple container name (“data”), it deletes the entire container.

Input Parameters

aduid	Agent Data Unit Identifier.
name	Name whose values are to be deleted from the ADU.

ADU.DeleteValues

IDL Syntax `ORBStatus DeleteValues( in ADU_ID aduid, in SeqString names) ;`

Description Given a list of names, this method removes from the ADU the names and their values, in effect making the ADU smaller. Values are deleted from active memory only. This method does not interact with the database.

If used with a container name (“a.b”), this method only deletes that one name, a.b. However, the ADU Server is not able to address names “below” that point. Although a.b.c may still exist, the ADU Server cannot find it, even though GetSubTree on “a” still sees them. Resolving a.b.c depends on resolving a.b, which does not exist anymore. Callers should generally *not* delete elements from non-leaf positions.

Input Parameters

aduid	Agent Data Unit Identifier.
names	List of names whose values are to be deleted from the ADU.

ADU.EventsIn

IDL Syntax `ONEWAY EventsIn(in string vdu_id, in SeqEvent events);`

Description This function adds a user-defined ADU event to an ADU. Values in the ADU are updated to reflect the names and values in the event.

Input Parameters

aduid	Agent Data Unit Identifier.
event	The event information. Each event must include an event name to describe the event. Requests without an event name are rejected.

Returns No return value. One way requests do not have any returns.

C Program Example

```
Event *event;

/* Create space for record */
event = vesp_event_create( "Update" );
vesp_event_add_couple( event, "name1", "value1" );

/* Create the new ADU History event entry */
status = Vesp_Request( " ADU.EventsIn", callback, 0x2132,
                          session, aduid, event );

/* release memory we allocated */
vesp_event_delete( event );
```

ADU.Find

IDL Syntax
```
ORBStatus Find( in string search_criteria, in unsigned long scope,
                out SeqString matches ) ;
```

Description This method returns a list of ADUIDs that match a simple criterion. The ADUs found may be active in the local ADU Server or in other WAN ADU Servers, depending on the setting in *scope*.

The criteria can be any expression an Assign method would accept.

The Find method, especially one distributed across the WAN, can take time, as it cannot reply until it receives a response from all the other servers it contacts and combines the lists it receives. A WAN-wide search proceeds in parallel across all involved ADU Servers, but the speed is still limited to the speed of the slowest single search. In the worst case, the request may start other servers that had not been started yet, which can entail a significant delay.

Input Parameters

search_criteria	Criteria to be used for the search, consisting of names and values. If values contain spaces or anything other than letters and numbers, they must be enclosed in double quotes. If the value contains a \ or " character it must be quoted by a \ character.
scope	Can be one of the following. Note that the "no_os" designation is for consistency with the EDU Server. ADU_NO_OS Check the local ADU Server only. ADU_NET_NO_OS Check the WAN ADU Servers but no Object Stores.

Output Parameters

matches	List of ADUIDs meeting the search criteria. Note that an empty list is considered valid and returns VESP_SUCCESS.

Returns

VESP_SUCCESS	Request was successful.

ADU.FindByKey

IDL Syntax `ORBStatus FindByKey( in string name, in string value, out ADU_ID aduid ) ;`

Description This method finds one active local ADU based on a single search criterion, or *key*. Each ADU contains at least one unique element, typically the login ID.

FindByKey() searches only the local ADU Server for active ADUs.

Input Parameters

name	Search key to find ADU.
value	Value to match if name is found.

Output Parameters

aduid	Agent Data Unit Identifier.

Returns

VESP_SUCCESS	Request was successful.

C Program Example

Locate an ADU having a key containing 1137.

```
status = Vesp_Request( "ADU.FindByKey", callback, 0x2132,
                          session, "key", "1137", &aduid );
```

ADU.FindOrCreate

IDL Syntax `ORBStatus FindOrCreate( in SeqCouple match, in SeqCouple, oncreate, out string vdu_id) ;`

Description This method searches the local server for an ADU that exactly matches all names and values in the match. It does not support wildcards. If a match is not found on the local server, it searches other ADU servers. If a match is found, it returns the id of the ADU. If multiple matches are found, the id of the most recently created ADU is returned.

If ADU.FindOrCreate does not find a match, the local server creates the ADU with the names and values from the combined list of match and oncreate. The id is then returned. This is the only case where the oncreate parameter is used.

Note: This method is intended to by used by the Avaya Toolkit. If you know the ADU needs to be created, it is recommended that you use the Create method because it is considerably faster and much more efficient.

Input Parameters

name	Search key to find ADU.
value	Value to match if name is found.

Output Parameters

aduid	Agent Data Unit Identifier.

Returns

VESP_SUCCESS	Request was successful..
VESP_ERROR	Requires values to find ADU. The match parameter may be empty.

ADU.ForceTerminate

IDL Syntax `ORBStatus ADU.ForceTerminate(in string duid)`

Description Forces the ADU object to be deleted from memory and from the database both, so it can no longer be used.

ADU.ForwardEvent

IDL Syntax `ORBStatus ForwardEvent( in unsigned long handleis, in Event Event) ;`

Description This method is reserved. ADU Servers use this method to pass events to each other. Client applications should not call this method.

ADU.GetActive

IDL Syntax

```
ORBStatus GetActive( out SeqADU_ID aduseq ) ;
```

Description This method finds all the active ADUs at the time the call is made. Note that only ADUs currently in memory are found.

Output Parameters

aduseq	A list of active ADUs.

Returns

VESP_SUCCESS	Request was successful.

C Program Example

```
SeqADU_ID ADU_ids;

/* Initialize a sequence of ADUs (0 specifies no limit) */
ADU_ids._maximum        = 0;
ADU_ids._length         = 0;
ADU_ids._buffer         = NULL;
status = Vesp_Request("ADU.GetActive", callback, user_data,
                      session, &ADU_ids );
```

ADU.GetOneValue

IDL Syntax

```
ORBStatus GetOneValue( in ADU_ID aduid, in string name, out string value ) ;
```

Description This method retrieves one value from an ADU.

Input Parameters

aduid	Agent Data Unit Identifier.
name	ADU element to be returned.

Output Parameters

value	One returned value from the ADU.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADUID or name not found.

C Program Example

Get the value of “myfavoriteelement” from the ADU named by aduid.

```
char *value;
status = Vesp_Request( "ADU.GetOneValue", callback, 0x2132, session,
                        aduid, "myfavoriteelement", &value );
```

ADU.GetSomeValues

IDL Syntax

```
ORBStatus GetSomeValues( in ADU_ID aduid, in string name,
                            out SeqCouple matches) ;
```

Description This function returns all names and values matching a container name. It is one of the few that accepts the * token. Thus, a *name* of data.*.emergency returns names such as data.1.emergency, data.brian.emergency, and so on.

Input Parameters

aduid	An ADUID.
name	A name. By intent, a container name.

Output Parameters

matches	All names and values in that container.

ADU.GetSubTree

IDL Syntax

```
ORBStatus GetSubTree( in ADU_ID aduid, in string name,
                        out SeqCouple matches) ;
```

Description This function returns all names and values in the subcontainer named by *name*. It can be given a container name (*ts*) or a subcontainer name (*ts.1*) and returns all names and values from that point down.

Input Parameters

aduid	An ADUID.
name	A name. By intent, a container or subcontainer name.

Output Parameters

matches	All names and values in the container or subcontainer.

ADU.GetValues

IDL Syntax `ORBStatus GetValues( in ADU_ID aduid, out SeqCouple values ) ;`

Description This method retrieves all of the values of an ADU.

Input Parameters

aduid	Agent Data Unit Identifier.

Output Parameters

values	A sequence to contain ADU information.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADUID was not found.

C Program Example

Get all of the data elements active in a specific ADU.

```
_IDL_SEQUENCE_Couple *values;

/* init an empty, unlimited sequence of Couples */
seq_couple = vesp_couple_seq_create();
status = Vesp_Request( "ADU.GetValues", callback, 0x2132,
                       session, aduid, seq_couple);
```

ADU.GetValueHistory

IDL Syntax

```
ORBStatus GetValueHistory( in string aduid, in string name,
                           out SeqString values, out SeqString when,
                           out SeqString who) ;
```

Description GetValueHistory returns everything that is known about the named field's values in an ADU. It reports on all the values the field has had (up to the limit specified with the *maxkeptrevisions* configuration parameter), who set them, and when.

The returned arrays are parallel, with the bottommost element being the most recent.

Input Parameters

aduid	An ADUID of an existing ADU.
name	The name of a field.

Output Parameters

values	A list of values to which the named field has been set.
when	When the field was set to each value, time_t string.
who	The session ID of the client that set each value.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADUID was not found.

ADU.GetValuesHistory

IDL Syntax

```
ORBStatus GetValuesHistory( in ADU_ID aduid, in unsigned long flags,
                            out SeqLong headers, out SeqString names,
                            out SeqSeqSeqString values) ;
```

Description GetValuesHistory returns everything that is known about all values in an ADU. For each named field in an ADU, it reports on all the values the field has had (up to the limit specified with the *maxkeptrevisions* configuration parameter), who set them, and when. By providing the various ADU_GSGET flags, you can control which of these values are returned.

The output in *headers* is a list of your specified flags, one per long integer. This indicates the order in which the bottommost sequence of strings in the values parameter are filled. Thus, if you specify (ADU_GSGETVALUE | ADU_GSGETTIME), headers would come back with two long integers, one containing ADU_GSGETVALUE and one containing ADU_GSGETTIME. The bottommost sequences in the values parameter would list values in the same order.

The output in *names* is a list of all the data names within the ADU. Container elements each get their own entry. The topmost sequences in values are in the name order, so if names[3] is “x”, values[3] is the sequence that describes the history for “x”.

Values itself runs three levels down. The topmost level is parallel to the *names* sequence. The next level down represents one entry for each value that *names* has had over time. If a value has only been set once (a common case) this sequence only has one element. The lowest level runs parallel to the headers parameter, and has one element for each of the kinds of data requested (any combination of text, time, and who).

For example, field *quark* was set twice, once at ADU creation (11:37:00am, by Scott, to “truth”) and again at ADU transfer (11:38:00, by Jane, to “charm”). Specifying flags = ADU_GSGETTIME | ADU_GSGETVALUE | ADU_GSGETWHO would yield:

```
header[0] = ADU_GSGETVALUE;          //first subelement is the text
header[1] = ADU_GSGETTIME;           //second is time
header[2] = ADU_GSGETWHO;            //third is who

names[?] = “quark”;                  //one of the names returned will be quark

values[?][0][0] = “truth”;           //values[13] is parallel to names[13], and the
                                     //first value that was set was “truth”
values[?][0][1] = “8145264302”;      //the time_t it was set (11:37:00am)
values[?][0][2] = “Scott”;           //Scott set it
values[?][1][0] = “charm”;           //next value set was “charm”
values[?][1][1] = “8145264362”;      //set at this time
values[?][1][2] = “Jane”;            //by this loginid
```

Input Parameters

aduid	An ADUID of an existing ADU.
flags	One or more of the ADU_GSGET flags, OR'd together. Flags are: #define ADU_GSGETVALUE 0x01 #define ADU_GSGETTIME 0x02 #define ADU_GSGETWHO 0x04

Output Parameters

headers	A list of ADU_GSGET values.
names	A list of ADU data names.
values	A list of data values (one per name), which are lists of value histories (one per value), which are lists of fields (text, when, and who).

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADUID was not found.

ADU.IncrValue

IDL Syntax

```
ORBStatus IncrValue( in ADU_ID aduid, in string name, in long incr,
                out string newvalue ) ;
```

Description This method is a useful alternative to using SetOneValue and GetOneValue to modify a value when there is a risk that two applications might conflict.

This method changes one ADU data element. If the element does not exist it is created. An element that exists is overwritten if permission allows. The value of the element is treated as a number and is converted to a long integer (by use of the C function atol). The specified increment is added, and the element's value is set to the result. Elements that do not exist are treated as if they contained a zero (0). The increment may be a negative value.

The **atol** function is robust in that if it cannot recognize a number, it returns 0. The method stores the resulting, incremented value as a decimal digit string without extraneous spaces or leading zeros. Thus, incrementing "x2" by one results in "1" and incrementing "00034E6" by one results in "35".

Input Parameters

aduid	Agent Data Unit Identifier.
name	The name of the ADU element to change or create.
incr	Increment value.

Output Parameters

newvalue	The new value of the element, after the increment.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADU not found, or the element cannot be changed.

C Program Example

```
char *result;
status = Vesp_Request( "ADU.IncrValue", callback, 0x2132, session,
                        aduid, "some_numeric_element", 1L, &result );
```

ADU.Monitor

IDL Syntax

```
ORBStatus Monitor( in string monitorcriteria ) ;
```

Description This method changes Assign criteria. It works like Assign in all respects. You must have previously assigned (and have not since deassigned) to use this method.

Input Parameters

monitorcriteria	Monitor criteria string. If values contain anything other than letters and numbers (for example, spaces), they should be enclosed in double quotes, and \ or " characters must be quoted by a \ character. Refer to "ADU Event Monitoring," on page 27 for additional information.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	Invalid monitor criteria.

ADU.RemoteWatcher

IDL Syntax `ORBStatus RemoteWatcher( in unsigned long handleis, in string cri) ;`

Description This method is reserved. Client applications should not call this method.

When multiple ADU Servers are listed in one directory, any Assign, Monitor, or Deassign method invocation is communicated by the local ADU Server to all other ADU Servers by a RemoteWatcher method invocation.

RemoteWatcher events are included in the ADU Server log file following Assign, Monitor, or Deassign events. They take the form:

uuid.RemoteWatcher(*nnnn*, "*abcde*")

where: *uuid* is the UUID of the ADU Server

nnnn is a number used internally.

abcde is a string identifying the Assign or Monitor criteria. A blank string indicates a Deassign method invocation.

ADU.SetAndTerminate

IDL Syntax `ORBStatus SetAndTerminate( in ADU_ID aduid, in SeqCouple data ) ;`

Description This method combines a SetValues and a Terminate.

Input Parameters

aduid	An ADUID.
data	Values to be set, not to exceed 1023 values.

ADU.SetAndTransfer

IDL Syntax

```
ORBStatus SetAndTransfer( in ADU_ID aduid, in string to,
                  in SeqCouple values ) ;
```

Description This method combines SetValues and Transfer into a single call, as these operations often occur together. If the operation succeeds, it generates a **Change** event containing all changes made to the ADU. This operation is usually performed by Avaya IC components, not client software.

Input Parameters

aduid	Agent Data Unit Identifier.
to	A string representing the receiving client.
values	A list of names and values to apply to the ADU, not to exceed 1023.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADU not found, set failed, or client not a valid string.

C Program Example

```
/*Transfer an ADU. */
char *to;
_IDL_SEQUENCE_Couple *values;

/* init an empty, unlimited sequence of Couples */
seq_couple = vesp_couple_seq_create();

/* fill in the field for the new ADU entry */
vesp_couple_seq_add_couple_values( seq_couple,"loginid",
                                   "meritha" );
to = ORB_object_to_string( VESP_ORB, &environment,
                           to_object);
status = Vesp_Request( "ADU.SetAndTransfer", callback, 0UL,
                       session, aduid, to, seq_couple );

vesp_couple_seq_delete( seq_couple );
```

ADU.SetDefaultHistoryFilter

IDL Syntax

```
void SetDefaultHistoryFilter( in unsigned long mask ) ;
```

Description This method is reserved. Client applications should not call this method.

This method allows the caller to specify which types of events are saved when ADUs are sent to the server named in the eventsink configuration parameter. The filter takes effect in subsequently created ADUs, not existing ones. All events generated for all subsequent ADUs are checked against the permissions in the bits set in the mask argument. Events already generated are not affected.

The SetHistoryFilter()method can be used to override the default filter for individual ADUs.

The bits are defined in the file **generic.idl** as:

HS_NOSTART	(unsigned short)0x01 Do not record new events
HS_NOCHANGE	(unsigned short)0x02 Do not record change events
HS_NOTRANSFER	(unsigned short)0x04 Do not record transfer events
HS_NOUSER	(unsigned short)0x08 Do not record user-defined events
HS_NODELETE	(unsigned short)0x10 Do not record delete events

End events (ADU.End) cannot be filtered out. They must be included in the stored ADUs. Data elements in the End event can be filtered with the **aduhskeepname** configuration parameter.

ADU.SetHistoryFilter

IDL Syntax

```
ORBStatus SetHistoryFilter( in ADU_ID aduid, in unsigned long mask ) ;
```

Description This method allows the caller to specify which types of events are saved when an individual ADU is sent to the server set with the **eventsink** configuration parameter, overriding the default history filter for one ADU. The filter takes effect immediately. All subsequent events generated are checked against the permissions in the bits set in the mask argument. Events already generated are not affected.

The bits are defined in the file **vespidl.idl as:**

HS_NOSTART	(unsigned short)0x01 Do not record new events
HS_NOCHANGE	(unsigned short)0x02 Do not record change events
HS_NOTRANSFER	(unsigned short)0x04 Do not record transfer events
HS_NOUSER	(unsigned short)0x08 Do not record user-defined events
HS_NODELETE	(unsigned short)0x10 Do not record delete events

The **end** event (ADU.End) cannot be filtered out. It must be included in the stored ADU. Data elements in the End event can be filtered with the **aduhskeepname** configuration parameter.

Input Parameters

aduid	Agent Data Unit Identifier.
mask	Permission bits to be set.

Returns

VESP_SUCCESS	Request was successful.

C Program Example

```
Vesp_Request_Sync(
"ADU.SetHistoryFilter",           /* method identification */
&ev,                              /* environment pointer */
session,                          /* session object */
&request,                         /* pntr to pntr to request structure */
aduid,                            /* an ADU id */
HS_NOUSER                         /* The permission bit to be set
                                     for this ADU */ );

/* check ev for errors as needed */
Vesp_Request_Delete(session, request);
```

ADU.SetOneValue

IDL Syntax `ORBStatus SetOneValue( in ADU_ID aduid, in string name, in string value ) ;`

Description This method sets one ADU data element. An element that does not exist is created. An element that exists is overwritten if permission allows.

The following fields are restricted and cannot be changed by applications:

aduid	termination
transfercount	duration
createtime	update_time
endtime	

Setting values can generate watch, change, and drop events.

Input Parameters

aduid	Agent Data Unit Identifier.
name	The name of the ADU element to change or create.
value	The value to be set.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADU not found, or the element cannot be changed.

C Program Example

```
/*Set one value. */
status = Vesp_Request( "ADU.SetOneValue", callback, 0x2132, session,
                           aduid, "my_favorite_element", "new value" );
```

ADU.SetValues

IDL Syntax `ORBStatus SetValues( in ADU_ID aduid, in SeqCouple values );`

Description This method sets one or more ADU data elements. Data elements that do not exist are created. Existing elements are updated if permission allows.

The following fields are restricted and cannot be changed by applications:

aduid	termination
transfercount	duration
createtime	update_time
endtime	

Setting values can generate **watch**, **change**, and **drop** events.

Input Parameters

aduid	Agent Data Unit Identifier.
values	A sequence containing ADU information, not to exceed 1023 values.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADU not found or values not able to be set.

C Program Example

```
/* Example of an initialized sequence with fixed values: */
status = Vesp_Request( "ADU.SetValues", callback, 0x2132,
                           session, aduid, &values );
```

ADU.SetValuesExtended

IDL Syntax
```
ORBStatus SetValuesExtended( in ADU_ID aduid, in SeqCouple data,
                        out SeqString newnames) ;
```

Description This function performs a SetValues method, but in addition returns a parallel list of names that it created while resolving container names. For example, if the *data* SeqCouple contained two names, agent.!.x and data.+, the *newnames* list might be returned containing agent.2.x and data.4.

This method is useful for applications that need to know how names were generated, especially for applications that make repeated use of the + token and need to be able to go back and fill in values in the various subcontainers they have created.

The following fields are restricted and cannot be changed by applications:

aduid	termination
transfercount	duration
createtime	update_time
endtime	

Setting values can generate watch, change, and drop events.

Input Parameters

aduid	An ADUID.
data	Values to be set, not to exceed 1023.

Output Parameters

newnames	List of names created to resolve container names.

ADU.Suspend

IDL Syntax `ORBStatus Suspend( in string aduid ) ;`

Description Like Terminate, this removes the caller's session from the ADU's session list, but it does not remove the loginid from the owner's list. Use this when your application is done with the ADU for now, but intends to do more work on it later. This method gives permission for the ADU server to retire the ADU to the DUStore, on the grounds that it may not be accessed for some time.

ADU.Terminate

IDL Syntax `ORBStatus Terminate( in ADU_ID aduid ) ;`

Description When a client creates, reads, modifies, or transfers an ADU, the client's name and session is added to internal lists. When the Terminate method is invoked, the client's name and session is removed from

the lists for that ADU. When the of names is empty, the ADU is purged from the system and is used for reporting. When the list of sessions is empty but the list of names is not (this can occur if Suspend is used), the ADU may be moved to the DUStore for temporary storage.

Input Parameters

aduid	Agent Data Unit Identifier.

Returns

VESP_SUCCESS	Request was successful.
VESP_PARTIAL_ SUCCESS	The ADU was not found in memory.
VESP_ERROR	Specified ADU does not exist.

C Program Example

```
status = Vesp_Request( "ADU.Terminate", callback, 0x2132, session,
                       aduid );
```

ADU.TerminateMine

IDL Syntax `void TerminateMine( ) ;`

Description When a client creates, reads, modifies, or transfers an ADU, the client's name is added to an internal list of clients. When the TerminateMine method is invoked, the client's name is removed from the list for *all* ADUs.

The Terminate() method is marginally faster than TerminateMine(), and is therefore preferable when the ADUID is known.

It is important to realize that transferring a call does not automatically end a client's responsibility toward the ADU. The client must use the ADU.Terminate() or the ADU.TerminateMine() method to signal that it has no further interest in the ADU.

Returns There is no return value.

C Program Example

```
status = Vesp_Request ( "ADU.TerminateMine" , callback, 0x2132, session );
```

ADU.Touch

IDL Syntax `ORBStatus Touch( in string aduid ) ;`

Description This method accesses an ADU, just as a GetOneValue might, but it modifies or fetches no values. As a result, the caller's session and loginid are placed on the internal lists for the ADU, and if the ADU was in the DUStore, it is brought into ADU server memory.

Input Parameters

aduid	Agent Data Unit Identifier.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADU not found in memory or in the ADU Server.

ADU.Transfer

IDL Syntax `ORBStatus Transfer( in ADU_ID aduid, in string to ) ;`

Description This method generates an ADU.Transfer event to any process monitoring the ADU, changes the value of the transfercount element and adds the new client to the ADU's list of interested parties. Transfers do not “move” an ADU anywhere.

Transferring an ADU does not end the client’s responsibility to the ADU. The client must use ADU.Terminate() to signal that it has no further interest in modifying the ADU.

Input Parameters

aduid	Agent Data Unit Identifier.
to	A string representing a new client.

Returns

VESP_SUCCESS	Request was successful.
VESP_ERROR	ADU not found, or client not a valid string.

C Program Example

```
/* Transfer an ADU */
char *to;
to = ORB_object_to_string(VESP_ORB, &environment, object);
Vesp_Request( "ADU.Transfer", callback, 0UL, session, aduid, to);
```

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y